Conflict Reduction in Hardware Transactions Using Advisory Locks

Lingxiang Xiang and Michael L. Scott

TRANSACT 2015 “lightning talk”

(full paper presented at SPAA this past Sunday)
Common pattern: transaction conflicts due to writes near the end:

**easy to conflict**

tbegin \[\text{beginning of transaction}\] tend \[\text{end of transaction}\]

- **thd1**
  - transaction starts early
  - conflict with **thd2** and **thd3**
  - abort

- **thd2**
  - transaction starts late
  - conflict with **thd1** and **thd3**
  - abort

- **thd3**
  - transaction starts late
  - conflict with **thd1** and **thd2**
  - abort
Common pattern: transaction conflicts due to writes near the end:

Suppose we could overlap the nonconflicting parts?
Staggered Transactions

- Compiler identifies instructions that are likely to be initial accesses to a shared cache block
  - inserts instrumentation for optional activation
- Runtime system collects statistics on causes of aborts
- Policy chooses which “advisory locking point” (ALP), if any, to activate in future, and which lock to acquire
  - may be based on instruction pointer or data address
  - may *promote* to “parent” IP based on *data structure analysis*
Hardware Support

1. To acquire an advisory lock, need nontransactional loads and stores
   – or transaction suspend/resume

2. On abort, want to know not only data address of conflict, but also PC of initial access
   – can get by with 12 bits/cache line—2.4% space OH
   – alternatively, can record these bits in ALP instrumentation
Simulation Results

- Speedup over HTM baseline with 16 threads
  - MARSSx86 simulator w/ AMD ASF
  - Staggered: 24% improvement (harmonic mean)